Projet d'Algorithmique 2
`Dedalus Explorer`: The nightmare continues...

François Goasdoué
ENSSAT – Université de Rennes 1

Le jeu `Dedalus Explorer`

`Dedalus Explorer` est un jeu de cartographie de labyrinthe où vous incarnez Thésée, héros de la mythologie grecque qui a trouvé, affronté et vaincu le Minotaure dans le Dédale. Vous pouvez vous rafraîchir la mémoire ici : `https://fr.wikipedia.org/wiki/Th%C3%A9s%C3%A9e`.

Dans ce jeu, vous devez concevoir et programmer l'Intelligence Artificielle de Thésée, c'est-à-dire sa stratégie de cartographie de labyrinthe.

Un niveau de jeu est codé de la façon suivante en ASCII :

```
**********
@.....****
*.*.*....*
*****.****
*...*....*
*.****.*.*
*......***
******...*
*......*..
**********
```

où
— '*' est un mur
— '.' est un chemin
— '@' est Thésée

À chaque tour de jeu, Thésée doit prendre la décision d'aller soit au Nord, soit à l'Est, soit au Sud, soit à l'Ouest, soit terminer son exploration. Pour cela, plusieurs informations sont disponibles : la cartographie de la portion déjà explorée du niveau, la position courante de Thésée sur cette carte, et dans quelle(s) direction(s) il est possible d'aller (en fonction de la présence de murs). Pour cela, le contrôleur de jeu vous demande uniquement le prochain mouvement à effectuer grâce à la fonction `theseus` que vous devez développer :

```
Move theseus(ExpTree map, ExpTree pos, bool north, bool east, bool south, bool west) {...}
```

En sortie de fonction, vous devez retourner le mouvement (Move) que vous jouez parmi les quatre possibilités North, East, South, West ; vous pouvez également stopper votre exploration en retournant None. Move est défini de la façon suivante :

```
enum compass {North, East, South, West, None};
typedef enum compass Move;
```

La cartographie de la portion déjà explorée du niveau, ainsi que la position de Thésée sur celle-ci, sont données par l'arbre d'exploration courant (map) et la position de Thésée dans cet arbre (pos) ; ils sont codés par des pointeurs sur noeud d'arbre quaternaire (ExpTree) :

```
struct Node {Move m; struct Node * north; struct Node * east; struct Node * south; struct Node * west;};
typedef struct Node * ExpTree;
```

Un noeud d'arbre d'exploration (Node) modélise quel mouvement a mené à ce noeud (m), et les sous-arbres d'exploration si on se dirige vers le nord (north), l'est (east), etc.

L'arbre d'exploration initial, lorsque vous êtes à l'entrée du labyrinthe, a pour mouvement None, et des sous-arbres vides (NULL) pour north, east, south et west.

L'affichage ci-dessous vous montre en mode débogage (cf. section suivante) les informations disponibles pendant que Thésée explore le niveau montré ci-dessus (l'adresse mémoire de chaque noeud est donnée pour retrouver où se trouve Thésée) :

```
Theseus move: S
Health: 99% (excellent)
Exploration rate: 28% (poor)
map: <0x7fd53c500640:None,0,<0x7fd53c5006d0:E,0,<0x7fd53c600040:E,0,<0x7fd53c500790:E,0,
<0x7fd53c600080:E,0,<0x7fd53c6000c0:E,0,0,<0x7fd53c500830:S,0,<0x7fd53c5007c0:E,0,
<0x7fd53c700000:E,0,<0x7fd53c6000f0:E,0,0,0,0>,0,0>,0,0>,<0x7fd53c600000:S,0,0,
<0x7fd53c5007f0:S,0,0,0,0>,0>,0>,0>,0,0>,0,0>,0,0>,0,0>,0,0>,0,0>
pos: 0x7fd53c5007f0
**********
......****
*.*.*....*
*****.****
*...*@...*
*.****.*.*
*......***
******...*
*......*..
**********
```

Votre objectif est de construire la meilleure Intelligence Artificielle possible afin que Thésée explore un maximum du labyrinthe en un minimum de mouvements.

Pour cela, vous aurez besoin d'extraire des chemins au sein de l'arbre d'exploration. Pour les représenter, vous **devez** utiliser une liste chaînée de movement codée par le type `string` déjà utilisé pour le fil d'Arianne dans le projet d'Algorithmique 1 :

```
struct link {Move m; struct link * next;};
typedef struct link * string;
```

Attention, dans ce projet, vous ne devez utiliser que des arbres `ExpTree` et des listes chaînées `string` : les tableaux de mouvements ne sont pas autorisés (= leur utilisation sera sanctionnée) !

Travail à réaliser en monôme

Récupérer les fichiers de travail sur l'ENT, en haut de la page du cours d'*Algorithmique 2* dans la section Projet. Ces fichiers sont dans Archive_Etudiants.zip ; pour celles et ceux qui utilisent leur ordinateur personnel avec la dernière version d'Ubuntu, utilisez Archive_Etudiants-Ubuntu-18.04-LTS.zip.

Ces archives contiennent un Makefile et trois répertoires :

1. `Dedalus_Explorer` contenant trois fichiers :
   - `dedalus_explorer-bw.o` : le moteur du jeu en noir et blanc,
   - `dedalus_explorer-color.o` : le moteur du jeu en couleur,
   - `dedalus_explorer-debug.o` : le moteur du jeu en mode débogage.
2. `Player` contenant deux fichiers :
   - `dedalus_explorer.h` : le fichier d'entête définissant les structures de données que vous aurez à manipuler dans `theseus_explorer.c`. CE FICHIER NE DOIT PAS ÊTRE MODIFIÉ. Le fichier original sera utilisé pour compiler votre fichier `theseus_explorer.c`.
   - `theseus_explorer.c` : le joueur aléatoire que vous devez modifier pour faire votre propre joueur. Pour cela, vous devez modifier la fonction `theseus`. Celle-ci pourra

faire appel à autant de sous-modules que nécessaires, ceux-ci devant être développés UNIQUEMENT dans le fichier `theseus_explorer.c`. Chaque module développé devra être **impérativement** commenté de la façon suivante : (i) avant chaque module, la stratégie de résolution du problème dévolu au module doit être décrite (càd. l'idée générale de ce que fait le module et comment il le fait), puis, (ii) dans le code du module, la mise en oeuvre de la stratégie de résolution doit être expliquée (càd. les détails du codage de la stratégie utilisée sont expliqués au fur et à mesure du code).

3. `Levels` contenant huit fichiers :
   — `level1`, ..., `level8` : des labyrinthes exemples sur lesquels tester votre joueur.

Vous pouvez compiler `Dedalus Explorer`, sous Linux uniquement, grâce au `Makefile` en tapant dans un terminal (là où est le `Makefile`) :

— `make` ou `make color` pour compiler le jeu en couleur.
— `make bw` pour compiler le jeu en noir et blanc.
— `make debug` pour compiler le jeu en mode débogage. Dans ce mode, l'écran n'est pas rafraîchit. Les mouvements de jeu sont affichées les uns à la suite des autres, ainsi que les informations disponibles (cf. exemple ci-dessus). Dans ce mode, vous pourrez voir les `printf` de votre code afin de faciliter la mise au point du programme.

L'exécutable généré par le `Makefile` se nomme `dedalus_explorer`. Pour lancer le jeu `Dedalus Explorer`, vous devez utiliser une commande de la forme :

```
./dedalus_explorer level_filename [maximum_number_of_moves game_rate]
```

où

— `level_filename` est le niveau sur lequel jouer (`level1`, ..., `level8` par exemple)
— et de façon optionnelle
   — `maximum_number_of_moves` est le nombre maximum de mouvements autorisés ;
   — `game_rate` est le temps entre deux movements du joueur en microsecondes.

Modalités de remise du travail

Votre projet devra être rendu **avant** le dimanche 27 janvier à 23h59 via l'ENT, dans le dépôt prévu pour votre groupe sur la page du cours d'*Algorithmique 2*.

Vous remettrez un rapport au format PDF décrivant la strategie suivie par votre IA de Theseus, ainsi que le fichier `theseus_explorer.c` implantant cette IA. Ces deux fichiers seront déposés sous la forme d'une archive ZIP `Nom.zip` où `Nom` est votre nom. Cette archive comprendra UNIQUEMENT un répertoire à votre nom contenant votre rapport et votre fichier `theseus_explorer.c`. N'oubliez pas non plus de mettre votre nom dans la constante globale `monome` du fichier `theseus_explorer.c`.

Votre code sera passé au détecteur de plagiat afin de le comparer à ceux de la promotion. Les codes plagieurs/plagiés auront 0/20.

Bon travail !